Fluent 流固耦合基础教程

Fluent 流固耦合基础教程（上）

作者 Seventy721， 2012 年 2 月

（本文版权属于作者，欢迎转载，但是请保留作者信息）

最近用 fluent UDF 做了个弹性梁的流固耦合问题。这里把经验跟大家分享一下。这个帖子主要是介绍基本操作和程序编写，算是扫盲贴；对于 CFD，有限元、梁理论、流固耦合原理不做过多探讨，请大家见谅。另外作者水平有限，时间精力也有限，写得不好也请见谅了。

作者假设读者熟悉 Fluent，对 CFD 的基本概念有所了解，具备一定的 C/Fortran 语言编程技能，并且了解有限元的基本理论和程序编写。不过即使背景有所欠缺也无妨，点是了解基本流程和操作方法。

1. 流固耦合问题的种类

流固耦合问题可以分为很多种。按照耦合程度分类，可以分为强耦合问题和弱耦合问题。但是这种分类方法有两个流行的版本。

第一版本根据流体和固体之间相互影响的程度来划分，在工业应用中谈得比较多。如果固体运动很小，对 流场的影响不大，则认为是弱耦合。比如金属管道内的水流引起的管壁运动，机翼的振颤等，可以属于这类。在计算机技术不够发达的时代，这类问题一般采用简化 的流场模型，如简单几何形状的理想流体，解析出由于固体运动而引起的流场作用力，然后再将这些作用力施加到固体控制方程上，从而得到附加质量，附加粘度， 附加刚度，以及其他非线性项。随着计算机技术的发展，对这类问题的直接数值模拟逐渐成为可能。如果固体变形或运动比较大，其对流场的影响不可忽略，则属于 强耦合。强耦合问题必须对流场和固体进行细致的计算，充分考虑固体变形对流场的影响。阀门开闭，血液通过血管瓣膜，旗帜在风中飘舞，等都属于这类问题。解 决这类问题需要依靠数值模拟，传统的简化模型很难应用在这类问题上。

第二个版本根据流体和固体求解的模式来划分，在学术领域用的比较多。强耦合问题被认为是流场和固体 变形必须同时求解的问题。带有渗流的多孔介质变形问题可以算这类问题。弱耦合问题则是流场和固体变形可以分开求解，但是二者的信息交换通过循环迭代完成。多数工程问题都可以算做这类问题。

这里我要讲的是第一类划分标准中的弱耦合问题，但所用的数值方法也可以应用到强耦合问题中，只是结构体的有限元方法需要采用大变形理论。按照第二种划分标准来说，这里解决的是流场和固体变形/运动分别求解，两场之间的相互作用通过迭代实现。

2. 流固耦合问题的数值模拟

既然是数值模拟，则需将系统方程在时域上离散。在每个时间步上分别解算流体域和固体域。固体的变形或位移导致流固边界的运动。这个运动以动网格的形势传递给流场，然后求解流场。流场的解则包括了这个运动所导致的反力。将这个反力反馈给固体，求解其下一时间步的响应。如**图 1** 所示。

VISIT…

LANZAROTE
Caliente.COM

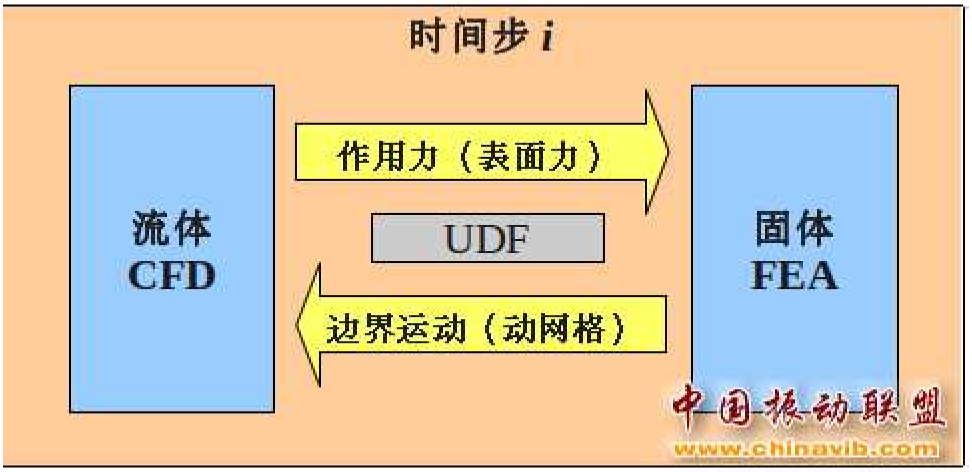

图 1

以上的概念恐怕大家都清楚，只是谈到实现的时候，需要做很多细致的工作。当然，如果大家有 ANSYS14 WorkBench，则流固耦合问题很容易解决，因为大部分的工作都自动化处理了。但是如果只用 Fluent，则需要做一些工作。即便使用先进的数值模拟工具，如果能弄清楚整个流程以及其中的一些技巧，也会对提高效率以及保证精度有所帮助。下面就讲讲具体怎么做。

3. 流固耦合例题

既然是具体讲，就不如找个例题来说。**图 2** 是一根柔软的弹性梁，浸泡在在直径为 Dt 的圆截面管道里。梁的截面也是圆形，直径为 D，长度为 L。梁的两端为铰接。整个结构是三维结构。梁可以在三维空间里弯曲。流体为水，梁中点所在截面上的平均流速为 U。管道入口处的流速为 Uin。上游长度为 Lup，下游长度为 Ldw。上下游的长度分别为 10D。

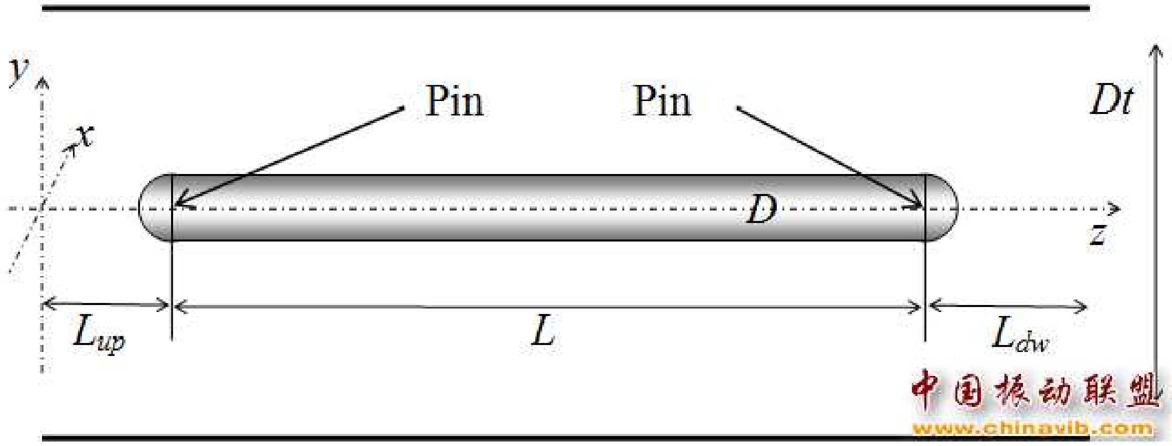

图 2

4. 流体模型

流体模型的建立是关键的一环，因为流体模型的好坏直接影响到所得解的合理性。建立好的

流体模型需要 利用流体力学知识对所处理的问题进行综合分析。根据流体的雷诺数和其它特征建立合适的网格，选择合理的湍流模型和算法。我们希望流体模型能够尽量准确地捕 捉到梁表面的力，因此准确的 near-wall 处理方法很重要。采用 wall function 还是精确地模拟边界层，需要给出适当的理由。由于我们处理的是动力问题，希望模型对梁表面力的变化给出准确的值。采用动态 k-epsilon 或者 k-omega 湍流模型计算所得的激励幅值明显偏小，这是由于雷诺平均造成的。这里我们采用对流体动力特性捕捉比较好的大涡湍流模型(Large Eddy Simulation, LES)。LES 对 near-wall 处理方法要求比较高，需要边界上的第一层网格厚度满足 y+ ~ 1。在 Gambit 里可以方便地建立边界层网格，达到以上要求。如何建立良好的网格是个复杂的话题，这里且不涉及。

由于固体-液体边界要移动，流体网格必须改变，需要运用动网格技术。此时建模的时候需要将动网格区域和不动网格区域分开，如**图 3** 所示。

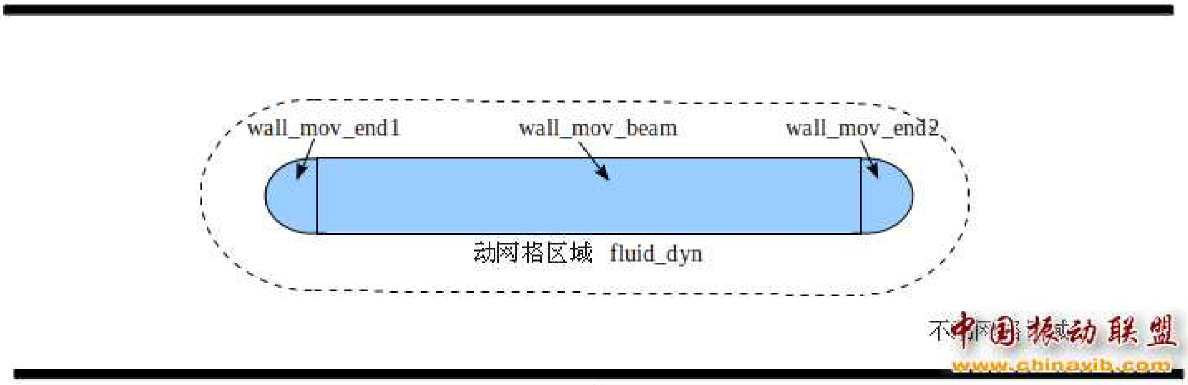

图 3

固体和液体的交界面附近的区域是动网格区域（fluid_dyn），远离固体的区域可以定义为不动网格区域。动网格区域的大小应当视具体问题而定，但起码要能够囊括固体的最大位置变化范围。梁表面的网格节点位置随着梁的运动而变化，因此需要将梁的表面单独定义为一个区域 wall_mov_beam。梁的两个端点也需要进行控制，因此也可以分别定义独立区域：wall_mov_end1 和 wall_mov_end2。在划分流体网格的时候，接近固体表面的地方需要建立边界层。Fluent 帮助文档上说边界层应该单独作为一个动网格区域，这是为了便于控制边界层的变形，防止出现边界层的异化。实际上如果将边界层的厚度增加到物理边界层厚度的两倍以上，并且只采用 smoothing 网格控制，则可以不必将边界层单独定义为一个区域。但是如果变形较大，流体网格需要重新划分，则最好将边界层单独分离为一个区域，并且在这个区域内使用 smoothing 控制。对于管壁附近的边界层，我们并不关心管壁上的压力和剪力，而且考虑到出了边界层以后这个区域的局部解对整体解影响不是很大，因此可以适当降低网格要求，甚至采用 wall function 近似逼近，这样可以节省不少计算量。下面两张图分别是**图 4** 几何模型（部分）和**图 5** 网格划分（部分）。

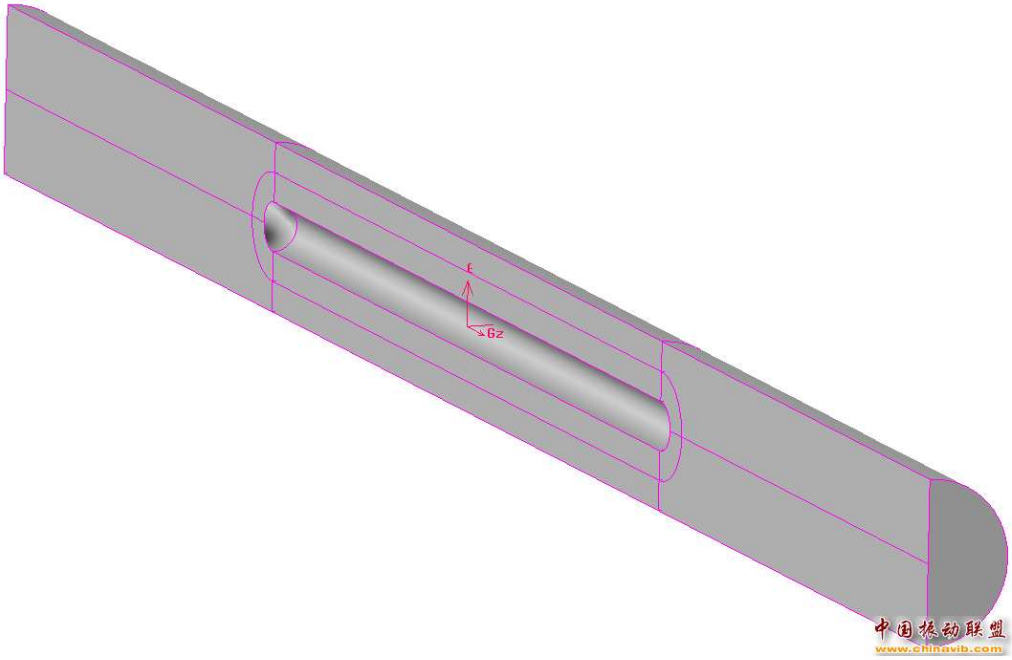

图 4

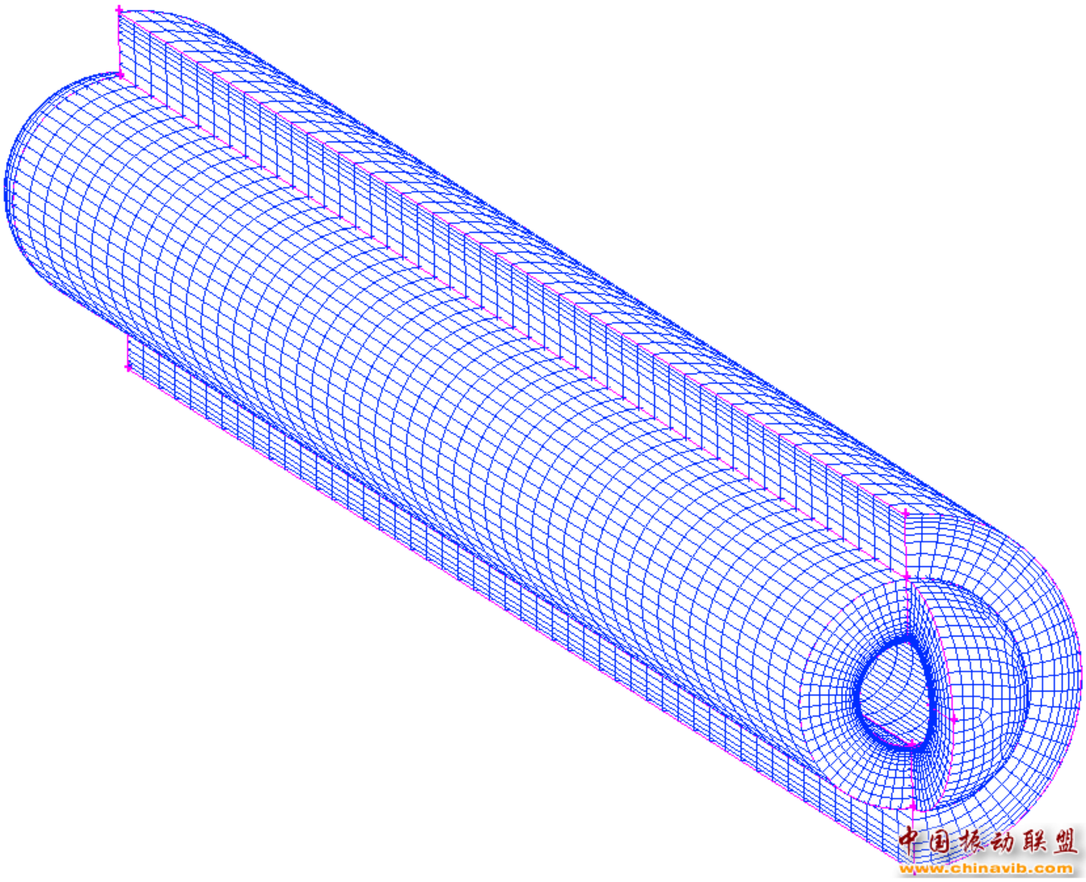

图 5

流体为不可压缩流体，求解器采用一次隐式瞬态算法。LES 的动网格计算上 Fluent 不支持二

次格式，这就需要时间步设定要小一些。时间步长和求解器参数的选取也是个复杂的问题，这里也不展开讨论了。在这个算例中，我们取较大的步长，以加快计算速度。具体参数如下：

Time step size: 0.0005 s
Solver settings: unsteady / pressure-based / 1st order implicit
Viscous model: LES / Smagorinsky-Lilly / no dynamic stress
Solution settings:
velocity-pressure coupling: SIMPLE
Relaxation factor: pressure 0.3 / momentum 0.7 / density 1.0 / body force 1.0
Discretization: pressure standard / momentumn bounded central difference

在不加入流固耦合的情况下，计算结果收敛很好，稳定以后每个时间步上的循环（iteration）为两步。每个时间步循环终止时的残差为：
continuity x-velocity y-velocity z-velocity
4.2946e-04 8.3873e-06 8.1793e-06 1.3922e-04

流体模型解算成功之后就需要考虑动网格和固体变形的问题了。梁表面流体网格节点的运动需要利用 UDF（User Defined Function）来控制。这将在下面的章节里说明。这里首先设置动网格的参数。

动网格区域 fluid_dyn 应该被设置为 Deforming，采用的网格控制方法为 Smoothing，但是对于复杂结构，可能需要 Remeshing。具体选项的含义请参见 Fluent 帮助文档中的用户手册第 11 章（Modeling Flows Using Sliding and Deforming Meshes）。这里只说明大体思路。这个区域的网格将会随着这个区域的边界（梁表面 wall_mov_beam）变化。我们只要控制 wall_mov_beam，则 Fluent 可以自动计算出这个区域内的流体网格的新位置。参数设置如**图 6** 所示。

图 6

面区域 wall_mov_end 1 和 wall_mov_end2 也设置为动网格区域，跟 fluid_dyn 类似，具体的网格变形由 Fluent 自动处理。这是由于我们的梁是两端铰接的，因此端面上的网格变化不大。对于其他的情况，比如悬臂梁，这两个面上的点应该跟据梁的运动用 UDF 来控制，不能交给 Fluent 自动处理。 具体设置参数如**图 7** 所示。

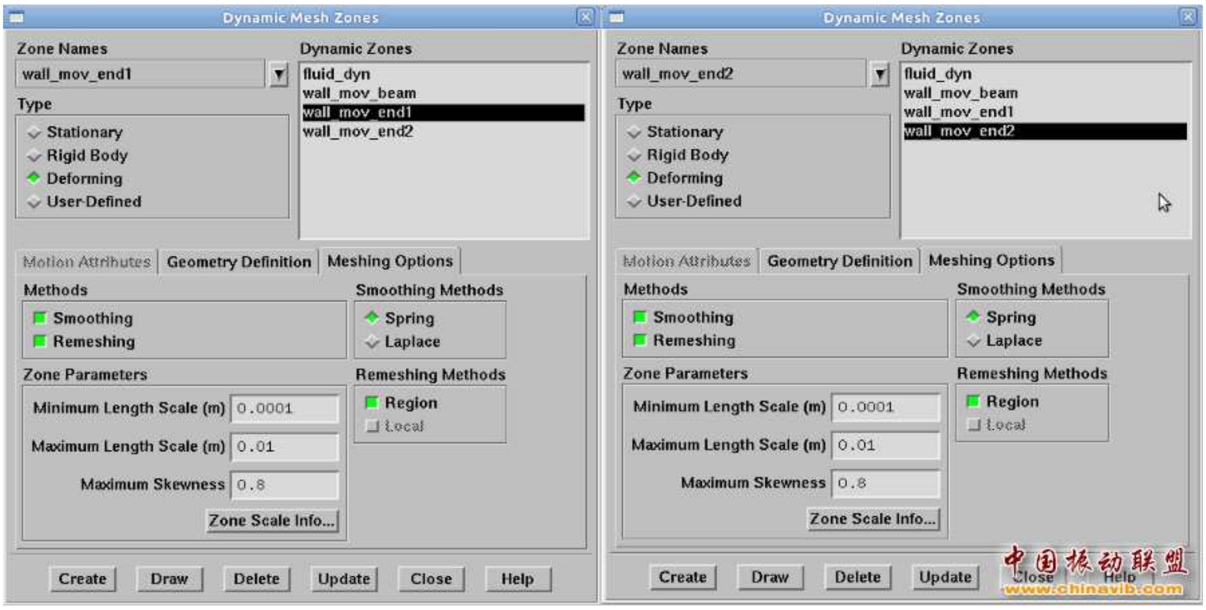

图 7

梁表面网格节点的位移必须通过 UDF 来控制。因此在选择的时候应当选 User-Defined。在 Motion Attributes 里面选择相应的 UDF。这一点在下面的章节中说明。这里先给出 Meshing Options 的值，参看**图 8**。这里提醒读者，wall_mov_beam 是驱动网格变化的“源头”。我们通过 UDF 控制 wall_mov_beam 的运动，而周边的网格将随着 wall_mov_beam 的运动而改变。

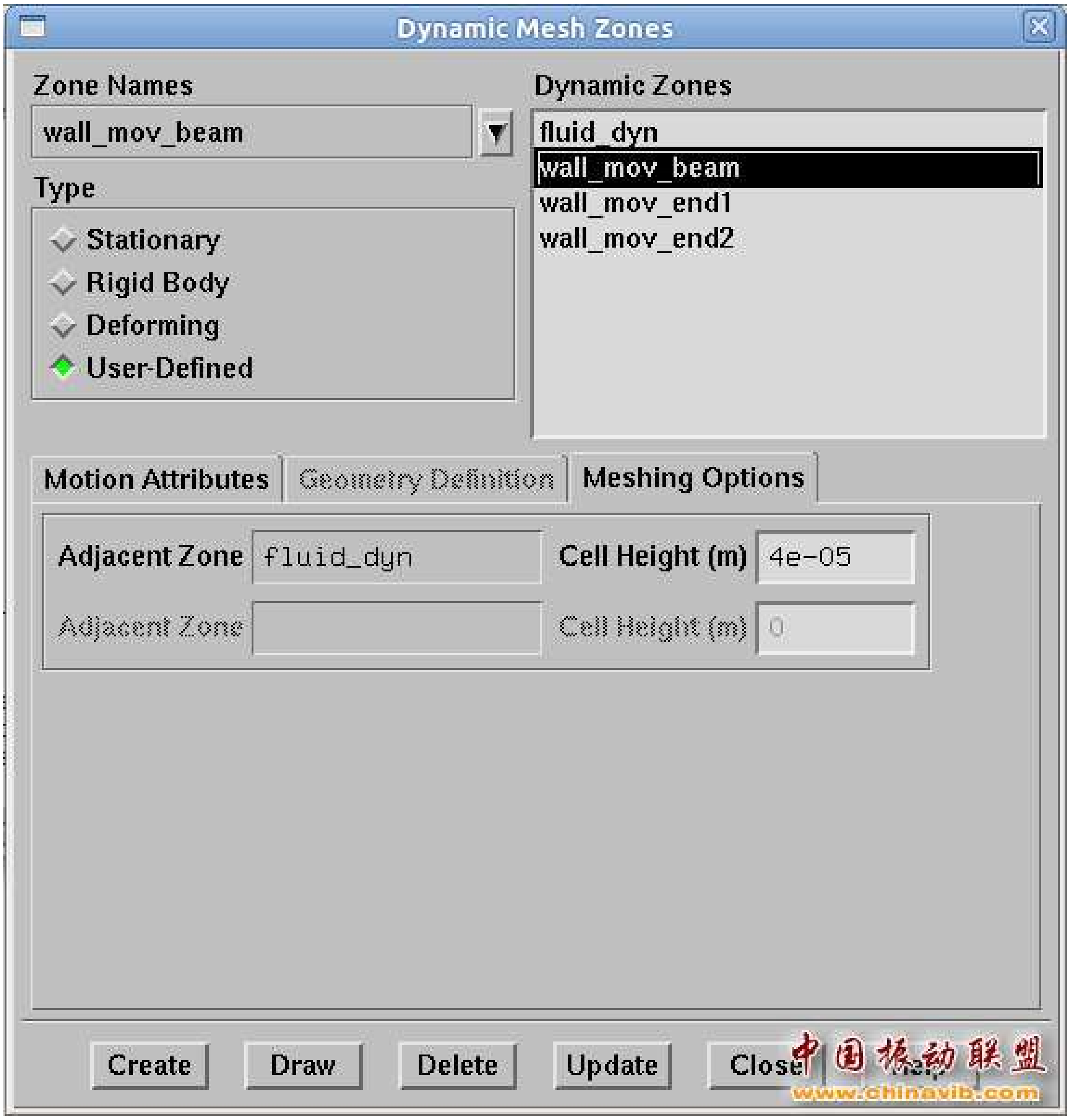

图 8

5. 固体模型

固体变形和运动的求解可以采用很多方法。对于简单的问题，可以采用解析解。目前的这个弹性梁的问题，就可以用采用解析解的方法。对于复杂结构，有限元是比 较常用的方法。建立有限元模型有两种方案。一种是采用三维实体单元完整地模拟出整个梁。这种方法的好处是流体和固体的接触面两边都是实际网格，一边是流体 网格，一边是有限元网格。边界上的位移，速度，受力等计算都比较简单，缺点是固体模型计算量大。第二种方法是采用结构体单元，如梁，板壳等。固体模型简 单，但是接触面上的处理稍微复杂一些。结构体单元在工程上应用得比较普遍。

为了说明如何利用结构体单元做流固耦合，这里采用三节点的梁单元。梁单元是一维单元，单元的几何形状只是一条中性线，没有体积。梁的变形是由中性线的

位移 和梁截面的转动描述的。在流体模型中，梁的体积是存在的。梁表面上的流体网格节点的位置需要由梁的变形确定。因此一个关键的步骤是根据梁的变形计算出梁表 面的流体网格节点的位置。

这里的梁采用小变形条件下的欧拉梁。根据欧拉梁理论，梁截面为刚性面且保持与中性线垂直。图 9 所示为一梁单元。梁表面上有一流体网格节点 A。点 A 所在的截面轴向坐标为 ξ 。则流体网格节点的位置可以表示为

$$\boldsymbol{u}_A = \boldsymbol{u}_O + [\Phi]\boldsymbol{r}$$

其中 **uA** 是 A 点的位移向量，**u0** 是 O 点的位移向量。**r** 是 O 点到 A 点的位置向量。**Φ** 是旋转矩阵。对于一般情况，**Φ** 矩阵的分量是欧拉转角的函数，在小变形小转动条件下，可以简化为 O 点处梁截面的转角的函数:

$$[\Phi] = \begin{bmatrix} 0 & -\theta_3 & \theta_2 \\ \theta_3 & 0 & -\theta_1 \\ -\theta_2 & \theta_1 & 0 \end{bmatrix}$$

O 点处的位移和梁截面的转角可以通过有限元的形函数求得。

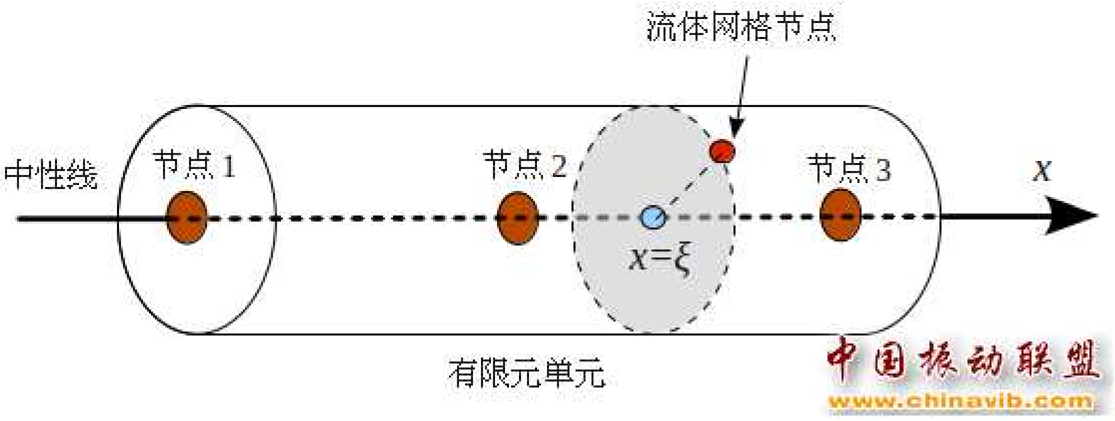

图 9

图 10 给出了梁变形后其表面上流体网格节点的位置。两个端点处的流体网格没有加以控制，而是交给 fluent 自动处理，这只是对当前这种支撑情况有效。对于一般情况，如悬臂梁，则需要人工计算。

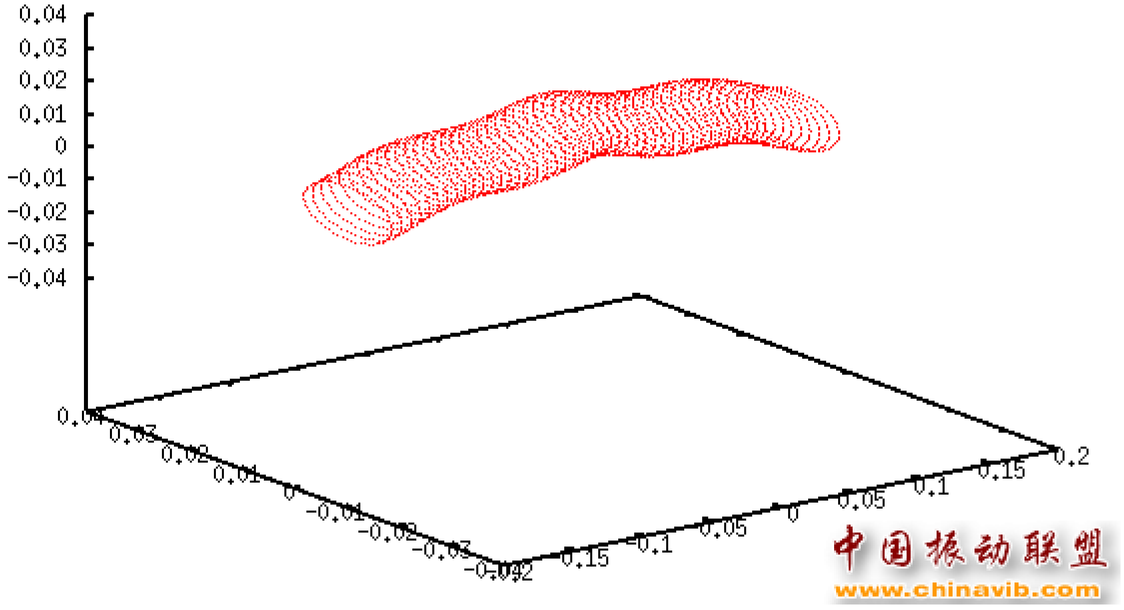

图 10

与网格位置计算类似。流体计算所得到的力需要传递给固体模型。梁表面的流体网格上的压力和剪力也需要利用有限元的形函数离散到有限元节点上。有限元方面的基本知识，我在这里就不罗嗦了。任何一本教材上都写得很清楚。

流固耦合问题是瞬态问题，因此需进行时间积分。常用的时间积分算法有 Newmark，Wilson-theta，Runge-Kutta 等。 Newmark 方法在多自由度的线性问题中应用比较普遍。这里我们也采用 Newmark 方法。但是在程序编写的时候需要注意 Fluent 求解流固耦合问题的流程。Fluent 必须作为整个过程的主导程序，如图 11 所示。

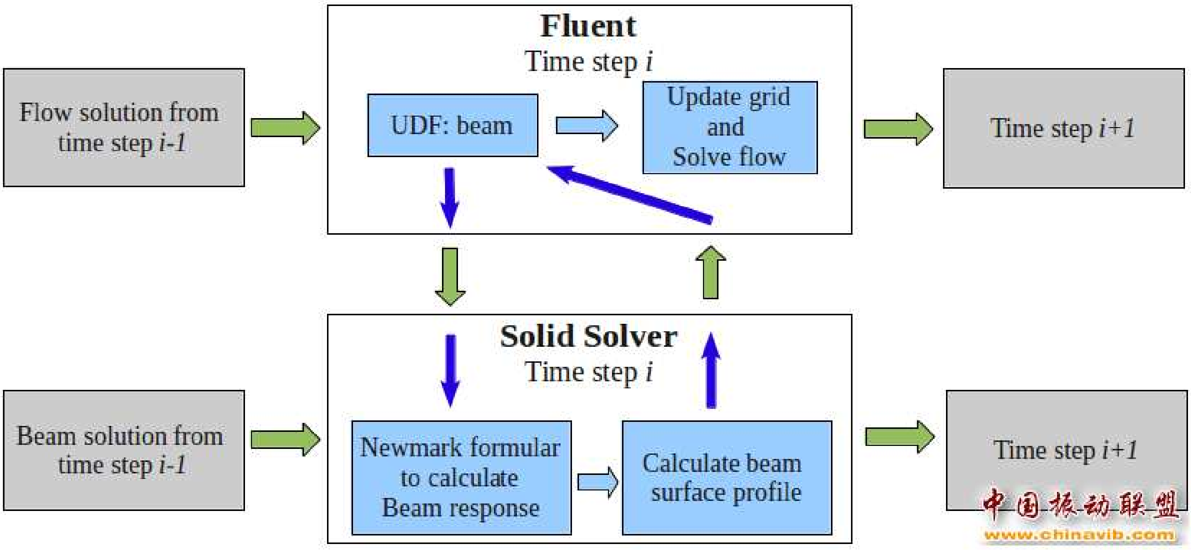

图 11

这种流程模式给程序编写带来一些麻烦，因为 Newmark 算法的循环被拆解开进行，必须按照单个时间步考虑。这就要求每个时间步之间的数据传递不能用通常 的变量存储。解决的办法有两个：一个办法是用硬盘存储，但是这样耽误在文件 I/O 上的时间很多；另外一个办法是利用常驻内存的 global 变量存储，但是 具体操作要看所用的编程语言环境。这里我用 Fortran90 编写的有限元程序，global 变量保存在独立的 module 里。如果只用 UDF 的话，跟 C 语言一致。

6. UDF

UDF 是用 Fluent 做流固耦合的关键。UDF 是 Fluent 用来提供可扩展功能的框架。做过 windows 或者 linux 程序开发的朋友会觉得很好理解，但不熟悉编程的朋友可能会觉得费解。在这里我简单解释一下它 的意思。Fluent 是个编译好的程序，想要实现功能扩展，最方便的办法就是利用动态链接库。在程序运行的时候将指定的动态链接库调入内存，然后通过预先定义好的接口函数执行 用户定义的子程序。换句话说 UDF 就是一些放在动态链接库里的函数。这些函数的名字和格式已经由 Fluent 预先定义好，但是内容是空的，需要用户来写。 Fluent 会在预定的情况下调用指定的函数，去运行用户定义的代码。所以对用户来说，就是填写一个指定的函数，编译成动态链接库，在 Fluent 里链接上，然后等着 Fluent 来调用。 如此简单。

这些预先指定好的函数由被称为定义宏（DEFINE macro）的宏命令来定义。这是个很拗口的说法，不过一般用户不必深究，拿它当函数来用就是。为了简单起见，下面用“UDF 函数”来代替“定义宏”。 Fluent 提供了很多 UDF 函数，具体有哪些，都是什么功能，诸位可以看 Fluent UDF 手册。这里就只说跟流固耦合有关的一个。下面的 UDF 函数用来定义网格的运动。

DEFINE_GRID_MOTION(name, d, dt, time, dtime)

除了编译的方法，Fluent 还支持对 UDF 的逐行解释执行。这样做方便调试，但是动网格的一些 UDF 函数是不能这样做的，所以我们还得用动态链接库。

UDF 是 C 语言编写的。Fluent 自带一个 C 编译器和一堆头文件。UDF 的编译可以在 Fluent GUI 里自动进行，但是也可以手工完成。有些情况下必须要手工操作，比如有 C/Fortran 混合编程的时候。顺便说一句，混合程序的编译也是个头疼的问 题，需要费很多周章。这跟所用的操作系统和 Fortran 版本有关。如果所写的 UDF 比较简单，只包括普通的 C 文件，则自动编译就可以。如果需要手工操作，则要建立如下文件结构

work folder （工作目录，模型放这里）
    |-> libudf （UDF 目录，有个 Makefile）
    |-> src （UDF 源文件放这里）
    |-> ultra_64 （这个可能根据所用操作系统有所不同）
    |-> 3ddp （3ddp 单机版）
    |-> 3ddp_host（3ddp 并行版主节点）
    |-> 3ddp_node（3ddp 并行版从节点）

然后修改 src/makefile 文件。之后回到 libudf 目录执行 make。

Fluent 的数据结构是 cell – face – node，如图 12 所示。每个网格的数据点是中心点 (cell center)。

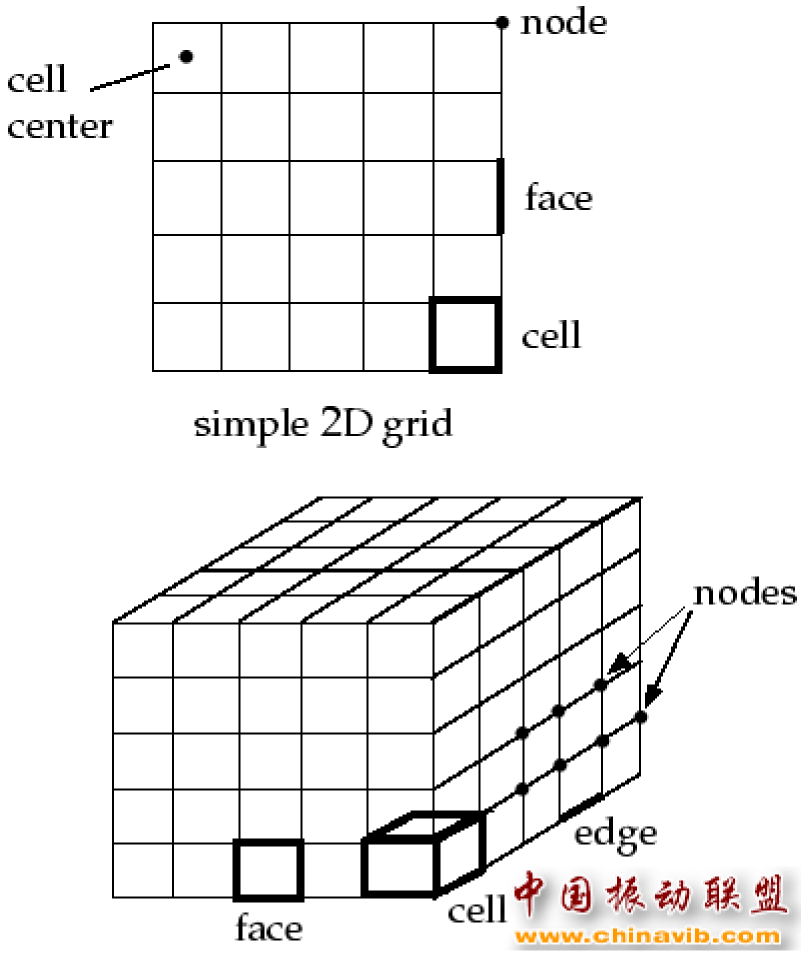

图 12

6.1 UDF - DEFINE_GRID_MOTION

这一节讲一下 DEFINE_GRID_MOTION 的大体结构。

```
// 首先引用几个头文件。这几个头文件在 Fluent 的 src 目录下。
#include "udf.h"
#include "dynamesh_tools.h"
#include "sg.h"

// 定义计算梁响应的有限元函数。此函数是 Fortran 函数。
extern void beam_response_(
    int *nfgrid,
    double (*fgrid)[ARR_LIMIT_FACE_BEAM][3],
    double (*force)[ARR_LIMIT_FACE_BEAM][3],
    double *gamm,
    double *beta,
```

```
    double *dt,
    double *t,
    int *run,
    int *ndgrid,
    double (*dgrid)[ARR_LIMIT_FACE_BEAM*NNODE_ON_FACE][3],
    double (*disp)[ARR_LIMIT_FACE_BEAM*NNODE_ON_FACE][3],
    double (*velo)[ARR_LIMIT_FACE_BEAM*NNODE_ON_FACE][3],
    int *info
);
```

// ... (定义其它全局变量和函数)

```
/*
----------------------------------------------------------------------
---- */
/*    ---------------------    macros    are    defined    below    here
--------------------- */
/*
----------------------------------------------------------------------
---- */

/* Macro for defining dynamic mesh udf for the beam */
DEFINE_GRID_MOTION(beam,domain,dt,time,dtime)
{
    // beam - UDF 的名字
    // domain - 当前的 domain 数据结构，存储了有关网格的信息，但不是网
格本身
    // dt - 指向网格数据结构（thead）的指针
    // time - 当前时刻
    // dtime - 时间步长

    // Define variables: pointers and utilities
    Thread *t = DT_THREAD(dt);
    cell_t c;
    face_t f;
    Node *node;
    int idNodeL; // local index of a node inside a face
    int countF; // number of faces
    int countN; // number of nodes
    int index;
    int i,j,k;
    int run;
    int id;
```

// 定义计算结构响应需要的变量

```
double xi; // the axial coordinate of a grid node
double area; // area
double pressure; // pressure
double distance; // distance between two points
double a_by_es; // A'A/(A'e)
double gamm=0.5;
double beta=0.25;
```

// 定义主/从节点间数据传递用的数组... （省略）

```
// Define constants
const double PI=3.14159265358979323846;
const double VISCOSITY = 0.001;
const double DENSITY = 1000.0;
const double TOLERANCE = 1e-15;
const double LOWERLIMIT = 1e-8;
```

// 向量初始化 ... (省略)

// 这一段用来取得梁表面的流体网格节点位置

```
#if !RP_HOST // SERIAL OR NODE
countF=0;
countN=0;
begin_f_loop(f,t) // 扫描当前 domain 的所有 face
{
  if PRINCIPAL_FACE_P(f,t)
  {
      countF++;
      if(countF>ARR_LIMIT_FACE_BEAM)
      {
          // 错误处理 ... (省略)
      }
      f_node_loop(f,t,idNodeL) // 扫描当前 face 的所有 node
      {
           node=F_NODE(f,t,idNodeL);
           // NODE_X, NODE_Y, NODE_Z 存储了节点坐标，但是注意 node
不是整数类型，实际上是联合变量。
           arrNode[countN][0]=NODE_X(node);
           arrNode[countN][1]=NODE_Y(node);
           arrNode[countN][2]=NODE_Z(node);
           countN++;
       }
  }
```

```
    }
    end_f_loop(f,t);
    #endif
```

// 将数据传输到主节点 ... (省略)

// 将相邻区域设置为动网格

```
    #if !RP_HOST // SERIAL OR NODE
    SET_DEFORMING_THREAD_FLAG(THREAD_T0(t));
    #endif
```

// 计算表面力（见 6.2 节）

// 将数据传输到主节点 ... (省略)

// 计算梁的响应（见 6.3 节）

// 将数据传输到计算节点 ... (省略)

// 更新网格 （见 6.4 节）

```
}
```

6.2 力的计算

梁表面所受到的激励分为法向力和切向力。法向力由表面压力引起；切向力就是剪力。梁的表面覆盖着流体网格的面(face)。数据的计算在面中点(face centroid)上进行。法向力的大小等于压力乘以网格面积。剪力等于剪应力乘以网格面积。剪应力等于粘度乘以速度的导数。速度的导数可以简单近似为为 cell centre velocity 除以 cell centre 到 wall 的距离。当然也可以采用更高阶的近似方法。需要注意的是在并行系统上，网格被分为多个区，每个区之间的交界面上的 face 会被重复计算。为了防止 这种情况发生，需要用 PRINCIPAL_FACE_P 判断是否为该 face 实际存在于当前的区里。

```
// Obtain the centroid location and the force on the centroids
index=0;
begin_f_loop(f,t)
{
    if PRINCIPAL_FACE_P(f,t)
    {
    // Save the face id
    arrFaceID[index]=f;
    // Obtain the centroid coordinates and save in arrCentr
    F_CENTROID(vecXf,f,t);
```

```
    arrCentr[index][0]=vecXf[0]; // x
    arrCentr[index][1]=vecXf[1]; // y
    arrCentr[index][2]=vecXf[2]; // z
    // Obtain the area vector and the area
    F_AREA(vecArea,f,t);
    area=sqrt(pow(vecArea[0],2)+pow(vecArea[1],2)+pow(vecArea[2],2));
    // Obtain the pressure and calculate the pressure force
    pressure=F_P(f,t);
    vecLift[0]=vecArea[0]*pressure;
    vecLift[1]=vecArea[1]*pressure;
    vecLift[2]=vecArea[2]*pressure;
    arrPForce[index][0]=vecLift[0];
    arrPForce[index][1]=vecLift[1];
    arrPForce[index][2]=vecLift[2];
    // Obtain the shear stress and calculate the viscous force
    // get the face parameters
    BOUNDARY_FACE_GEOMETRY(f,t,vecArea,distance,vecEs,a_by_es,vecDr0)
    if(distance < TOLERANCE) // distance is always positive
    {
      distance=LOWERLIMIT;
    }
    // -- get the centroid velocity of the cell
    vecVelo[0]=C_U(c,t);
    vecVelo[1]=C_V(c,t);
    vecVelo[2]=C_W(c,t);
    // -- calculate the viscous force (= shear stress * area)
    vecDrag[0]=(VISCOSITY/distance)*vecVelo[0]*area;
    vecDrag[1]=(VISCOSITY/distance)*vecVelo[1]*area;
    vecDrag[2]=(VISCOSITY/distance)*vecVelo[2]*area;
    arrVForce[index][0]=vecDrag[0];
    arrVForce[index][1]=vecDrag[1];
    arrVForce[index][2]=vecDrag[2];
    // Increase index
    index=index+1;
    }
}
end_f_loop(f,t);
// Calculate the total force
for(i=1;i<=countF;i++)
{
    arrForce[1] = arrPForce[1] + arrVForce[1];
    arrForce[2] = arrPForce[2] + arrVForce[2];
    arrForce[3] = arrPForce[3] + arrVForce[3];
}
```

6.3 结构响应

结构响应由 Fortran 函数求得，采用 Newmark 时间积分。注意 Fortran 函数的末尾要加上一个额外的下划线。另外变量名前要加&，因为 Fortran 函数参数都是地址传递而非值传递。

```
// Perform the Newmark scheme.
if (flagInitial!=1)
{
    // Calculate the beam response
    info=0;
    for (i=1;i<=countN;i++)
    {
      arrDisp[0]=0.0;
      arrDisp[1]=0.0;
      arrDisp[2]=0.0;
      arrVelo[0]=0.0;
      arrVelo[1]=0.0;
      arrVelofor(i=1;i<=countF;i++)[2]=0.0;
    }
    run=1;
    beam_response_(
    &countF, // nfgrid,
    &arrCentr, // fgrid,
    &arrForce,              // force,
    &gamm, // gamma=0.5,
    &beta, // beta=0.25,
    &dtime, // dt,
    &time, // t,
    &run,                // run, 0-preparation only, 1-run
    &countN, // ndgrid,
    &arrNode, // dgrid,
    &arrDisp, // disp,
    &arrVelo,              // velo,
    &info // info
    );
}
```

6.4 网格更新

得到结构响应以后，需要更新梁表面流体网格节点的位置。可以采用 NV_V 命令赋值。NODE_COORD(node)对应的是节点 node 的坐标。由于节 点循环的时候会重复访问两个单元共有的节点，因此更新的时候需要用 NODE_POS_NEED_UPDATE 检查当前节点是否已经被更新过。

```
if (flagInitial!=1)
{
    // Update the grid nodes
    index=0;
    begin_f_loop(f,t)
    {
      if PRINCIPAL_FACE_P(f,t)
      {
          f_node_loop(f,t,idNodeL)
          {
              node=F_NODE(f,t,idNodeL); NV_V(NODE_COORD(node)
              // Update node if the current node has not been previously
              // visited when looping through previous faces
              if (NODE_POS_N{
    arrForce[1] = arrPForce[1] + arrVForce[1];
    arrForce[2] = arrPForce[2] + arrVForce[2];
    arrForce[3] = arrPForce[3] + arrVForce[3];
}
```

7. 计算结果

为了保证模型的正确性，需要查看数值模拟的结果。这里不讲如何诠释结果，只讲如何检查流固耦合过程是否正确。最直接的方法是观察网格的变形。简单的做法是 在 Write/Autosave 中定义自动保存的频率。一般按照时间步来保存，比如选择每 1000 个时间步保存一次。然后查看每次保存的时刻上网格的变形 程度。保存的时候一定要将 case 和 data 文件都保存下来。另外保存频率的选择要保证一个振动周期内保存至少 5 次以上。这样才能看出周期运动。图 13 是 结构网格变形的例子。

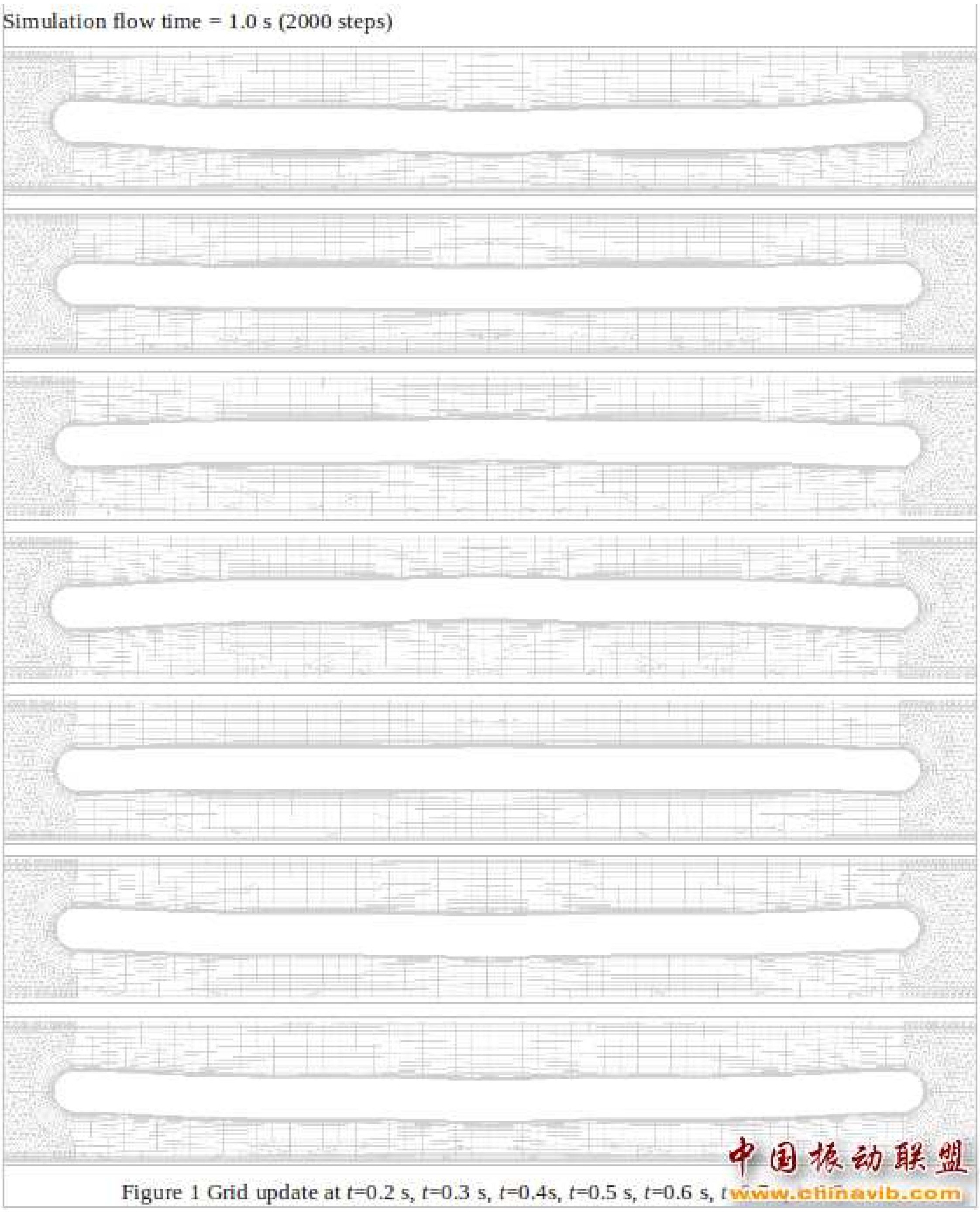

图 13

查看流体网格是比较麻烦的工作，但也是必须的工作。如果只查看固体变形的过程，并不能说明流场网格也正确变化。笔者第开始运行程序的时候就遇到这样的情 况。；流体网格没有更新，但是流体力被正确地传递给，而固体振动也是正常的。这样的结果得到是接耦合的流致振动解，比如湍流激励引起的振动。但是

这样的解 是不正确的，因为没有包括附加质量，流体阻尼，和附加刚度。所以一个很重要的方法是观察结构的响应，然后根据响应分析这些附加质量等效果是否存在。对于这 个梁，没有附加质量的自由振动周期为 0.4 秒左右，带有附加质量以后上升到 0.57 秒左右。与理论估计得到的结果一致。

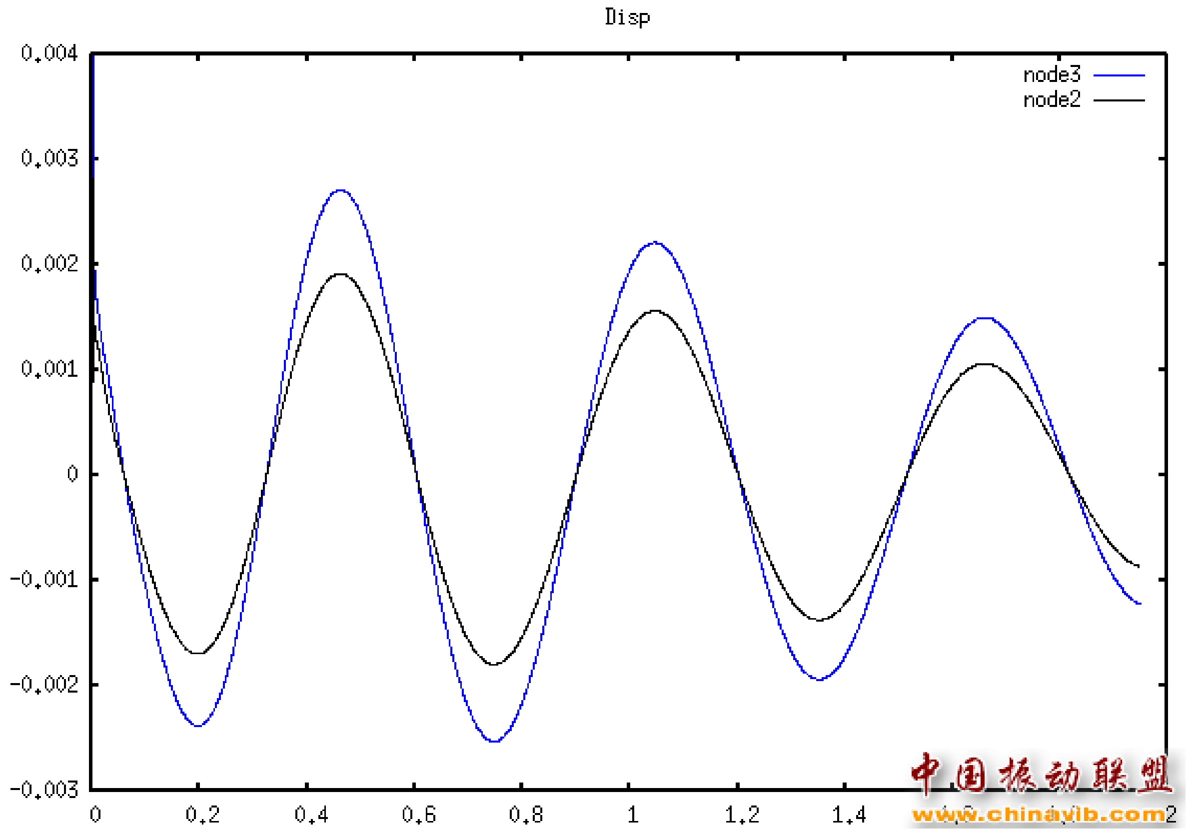

图 14

8. 后记

用每天挤牙膏的方式，终于写完了。水平有限，时间有限，只能写到这个层次了。这篇文章写得很粗略，只是作为大概的介绍，很多细致的内容还要读者自己去探索。流固耦合还是个很有意思的话题的。现在 ANSYS 做得好，恐怕大家以后都不用这么费劲写 UDF 了。耦合问题现在主要需要面对的是计算量问题。解决实际 工程问题，没有并行计算恐怕很难完成。

有限体积法和有限元已经都是很成熟的东西。关键是湍流问题不好解决。大涡理论是个很好的方向，只是 near wall 条件不好处理。如果能做出好的 wall function 就可以极大降低计算量。流固耦合的难题还是在强耦合问题上，据说 ADINA 做得不错。

希望这篇文章对打算进入流固耦合领域的朋友有点启发，起码算是个快速入门。这篇文章里的代码是具有代表性的段落，但不是完整的程序，请不要不加思考地照 搬。请不要找我要完整代码，我是不会给你的。做这个东西，我是有钱挣的。坦白地说一共挣了八千。如果你非要完整代码也可以，我给你打到极限折，只收十分之 一。顺便说一句，货币单位是美元。

朋友们好好学，这年头知识就是钱。